GHDLの使い勝手を試す

デバイスの記事

竹本 悟

フリーで使えるLinux対応VHDLシミュレータ

ここでは，フリー・ソフトウェアとして提供されているVHDLシミュレータ「GHDL」を評価する．VHDL2002対応を目ざして開発が進められているため，現在はベータ版である．VHDL87には対応しており，FPGA設計用や学習用としては十分な機能を備えている．（編集部）

VHDLが発表された当初，その規格が公開され，多くの大学で使われたことから，フリーのシミュレータが登場するのはそう遠くないと思われていました．事実，いろいろなフリー・シミュレータが登場してきました．しかし，残念なことに，ほとんどがVHDL規格のサブセット版でした．

最初のフルセットVHDLシミュレータは「savant」です．しかし，これば“純粋な”シミュレータでした．GUI（graphical user interface）や波形を表示する機能がなく，ユーザが波形を表示する方法を考えなければならなかったのです．また，構成が複雑で，だれもが簡単に使える代物ではありませんでした．

FreeHDLというプロジェクトがあります．ここではLinux版のVHDLシミュレータの開発が行われています．仕様書は存在していますが，まだ実用に耐えるレベルに到達していません．

このような中，突然フランスで登場したのが「GHDL」です．

GHDLの概要

GHDLはTristan Gingold氏が開発したVHDLシミュレータです．GNUのGPLに基づいて配付されています．したがって，基本的には無償，無保証です．

最初のバージョンが登場したのは2002年です．0.1から始まったバージョンは，本稿執筆時点で0.20になりました．まだベータ版という位置づけで，いくつかの構文チェックに不備な点が残っているようです．しかし，ほぼフルセットに近い機能を持っています．

● VHDL87に対応

GHDLは，VHDL87（IEEE 1076-1987）に対応しています．VHDL2002の仕様を満足させるのがバージョン1.0の目標だそうですから，まだまだアップデートは続きそうです．履歴を見ると，ほぼ1～2ヵ月間隔でバージョンアップが行われています．

添付の標準ライブラリも商用のライブラリと同じものが備わっています．とくに算術演算ライブラリは古いものと新しいものを備え，ユーザがオプションで選択できるようになっています．ドキュメントには古い算術演算ライブラリの持つ問題が指摘されており，使用上の注意も書いてあります．

VITAL（VHDL Initiative toward ASIC Libraries）のサポートもあり，sdfのバックアノテーションの機能も包含しています．

GUIは含まれていません．そのため，シミュレーション結果を波形で観測する際には，波形表示ツールが必要になります．

● 実行可能コードを生成するコンパイラ

多くのHDLシミュレータと違い，GHDLはコンパイラです．VHDLのソース・コードから実行可能なバイナリ・コードを作ります．

VHDLは米国国防総省のお声掛かりとあって，その仕様はPascalやAdaによく似ています．当然PascalやAdaを使ってコンパイラを作れば，比較的簡単に実現できそうな

GHDLのインストール

COLUMN

● **バイナリ・ファイルを使う場合**

まず，バイナリ・ファイルをダウンロードし，解凍します．解凍されたディレクトリの中にバイナリのtarファイルがあります．それをrootで/の位置で解凍すると自動的に/usr/localの下に必要なファイルがインストールされます．

```
cd /
su
tar xvf ghdl-0.20-i686-pc-linux.tar
```

後はパスが/usr/local/binに切ってあれば，すぐにGHDLを起動できます．

● **ソース・ファイルを使う場合**

ソース・ファイルからのインストールはちょっと複雑です．まず，AdaのコンパイラであるGNATをインストールします．次にGCCのソース・コードを入手します．GHDLのバージョンによって，要求されるGCCのバージョンが異なるので注意が必要です．

GCCのソース・コードの中にGHDLのソース・コードを組み込みます．そして`--enable-languages=vhdl`のオプションを付けてconfigureを実行します．

```
make CFLAGS="-O"
```

を実行させてコンパイルし，

```
make install
```

でGHDLコンパイラがインストールされます．

ここでは簡単にまとめましたが，実際にはけっこう複雑な作業になるため，できるだけバイナリを使うほうがよいでしょう．

気がします．GHDLはAdaを使ってVHDLのコンパイラを実現しています．そして，オブジェクト・コードを実行することでシミュレーションを行っています．

● **Linuxのみのサポートだが Windows 対応にも期待**

GHDLは，GHDLのWebサイト（http://ghdl.free.fr/）から入手できます．Linux版のソース・コードとバイナリ・コードが提供されています．

ソース・コードがあるので，ほかのOSへの移植も可能でしょうが，開発者によるサポートの予定はないようです．ただし，IVIのプロジェクトではGHDLのサポートを表明しています．すでにMacOSへの移植も試みられており，Solaris版やWindows版も期待されています．このプロジェクトでは，Verilog HDLシミュレータ「icarus」もサポートしているため，同じ環境でフリーのVerilog HDLシミュレータとVHDLシミュレータが動作することになります．

● **バイナリを使ってインストール**

筆者はソース・コードが要求するGCCのバージョンが合致しないのでバイナリを使用しました．バイナリを使うとインストールが非常に簡単です（上掲のコラム「GHDLのインストール」を参照）．

GHDLをソース・コードからコンパイルすることもできますが，非常にわかりにくい作業です．GCCとAdaを複合して作られているようで，GCCのほかにAdaのコンパイラであるGNATが必要です．コンパイル作業にある程度の経験がないと難しそうです．

なお，GHDLのGはマニュアルによると，特別な意味を持たないそうです．

GHDLを使ってみる

GHDLそのものの構造は非常にシンプルです．Cコンパイラと同じように，VHDLソース・コードを実行可能なオブジェクト・コードに変換します．シミュレーションは，そのオブジェクトを実行することによって実現されます．

ほかのプログラム言語と同じように，GHDLはVHDLソース・コードをコンパイルし，リンクして実行モジュールを作り出します．VHDLの世界では，これらを「アナライズ」，「エラボレート」と呼んでおり，GHDLのコマンドもその呼び名で説明されています．

VHDLソース・コードから実行モジュールを作り出す方法は，ソース・ファイルが一つの場合と複数の場合で異なります．

● **ファイルが一つだけの場合**

実際の設計では，ソース・ファイルが一つで済むケースなどはまれですが，GHDLの動作を理解するうえでは役に立ちます．

まず，アナライズを行います（**図1**の①）．評価には，



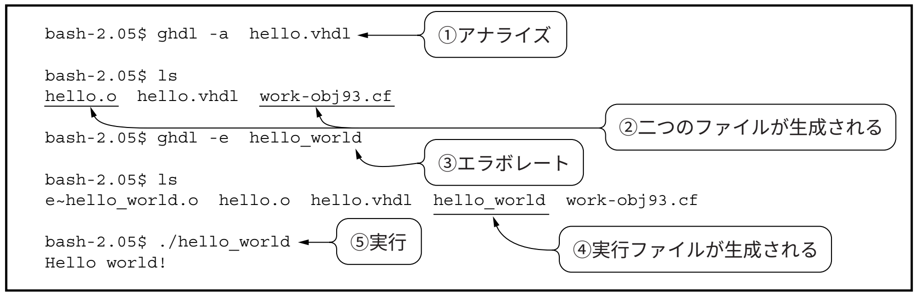


図1
ソース・ファイルが一つの場合の手順

GHDLのマニュアルにあるHello worldの記述を使いました．これをhello.vhdlというファイル名で保存します．アナライズは以下のコマンドで行います．

```
ghdl -a  hello.vhdl
```

-aはアナライズ命令を示しています．エラーがなければそのままプロンプトが返ってきます．そして，二つのファイルが生成されます(**図1**の②)．

次に，エラボレートを行います(**図1**の③)．エラボレートは実行可能なモジュールを生成します．

```
ghdl -e  hello_world
```

-eはエラボレーション命令を示しています．エンティティ名と同じ名まえの実行ファイルが生成されます(**図1**の④)．これを実行すると，“Hello world!”と表示されます(**図1**の⑤)．

実行には，ラン命令(-r)を使う方法もあります．シミュレーションにオプションを与えるときは，この方法を使います．

```
ghdl -r  hello_world
```

● 複数のソース・ファイルがある場合

複数のVHDLソース・ファイルをコンパイルする場合，コンパイルの順序が問題になります．とくに多数のファイルで構成される場合には，この管理がめんどうです．

この問題に対して，GHDLはインポートという便利なコマンドを用意しています．VHDLソース・ファイルを読み込んで，自動的にコンパイルの順序を決めてくれます．

ここでは，「DLX」というフリーのCPUコアを例に実行してみます．DLXは，オーストラリアのPeter J. Ashenden氏が教育用に作った32ビットのRISCプロセッサ・コアです．GHDLのWebサイトからソース・コードをダウンロードできます．

```
tar xvfz dlx.tar.gz
```

で展開すると，dlxディレクトリが生成されます．すべてのVHDLソース・コードはここに収納されています．まず作業用ディレクトリworkを作っておきましょう(**図2**の①)．

すべてのソース・ファイルを用意したら，インポート・コマンドを実行します(**図2**の②)．

```
ghdl -i ?workdir=work  *.vhdl
```

GHDLは拡張子vhdlを持ったすべてのファイルをスキャンし，アナライズの順序を決定します．そして実行ファイルを作るためメイク・コマンドを実行します(**図2**の③)．

```
ghdl -m --workdir＝work dlx_test_
behaviour
```

dlx_test_behaviourは最上位階層であるテストベンチのエンティティの名まえです．実行可能ファイルはこれと同じ名まえで，現在のディレクトリに作られます．

このファイルを実行すると，テストベンチの記述に従ってデバッグ情報を出力しながらシミュレーションが実行されます(**図2**の④)．途中で表示が止まりますが，シミュレーションは進行しています．そのままにしておくと無限に進行してしまうので，［Ctrl］＋［C］ボタンで停止します．

このままの状態では，バッチ・モードで実行するには不便です．シミュレーションを自動的に止まるようにするには，

```
ghdl -r dlx_test_behaviour
--assert-level=note
```

とします．ここでは，assert文のseverityレベルがnoteのときシミュレーションを停止することを指示して

```
bash-2.05$ cd dlx

bash-2.05$ mkdir work

bash-2.05$ ghdl -i ?workdir=work  *.vhdl

bash-2.05$ ghdl -m --workdir=work dlx_test_behaviour

bash-2.05$ ./dlx_test_behaviour
..........................
proc: end of execution
proc: fetching instruction...
bus monitor: I-fetch from X"00000010"
bus monitor:   X"1420FFF0" [ BNEZ     R1, -16 ]
proc: incrementing PC...
proc: decoding instruction...
proc: executing instruction...
proc: end of execution
proc: fetching instruction...
bus monitor: I-fetch from X"00000014"
bus monitor:   X"44000000" [ TRAP     0 ]
proc: incrementing PC...
proc: decoding instruction...
proc: executing instruction...
dlx-behaviour.vhdl:395:11:(assertion note): TRAP instruction encountered, execution halted
```

①ファイルの準備

②インクルード

③メイク・コマンドの実行

④実行．bash-2.05$ ghdl -r dlx_test_behaviourでもよい

図2　ソース・ファイルが複数ある場合の手順

います．

● std_logicの信号が観測対象

ここまでの説明を読んでも，シミュレーションを実施しているという実感をあまり感じられないのではないでしょうか？

現状，GHDLそのものには波形を表示する機能はありません．シミュレーション時にVCDファイルを出力し，VCDファイルを読み込むことのできる波形ビューワで表示することになります．

シミュレーション時にVCDファイルを作るためにはオプションが必要です．

```
ghdl -r dlx_test_behaviour ?assert-
level=note -vcd=wave.vcd
```

これはシミュレーション時に，波形をwave.vcdというファイルに保存するオプションの例です．GHDLではどの信号を保存するかを指示することができません．またどのような信号の波形でも保存できるわけではありません．VCDファイルに保存できるのは，`bit`か`std_logic`に限られます．詳細を調べたわけではありませんが，GHDLは記録できそうな信号だけを集めて，VCDファイルを作り上げているようです．観測したい信号があるのならば，あらかじめ`std_logic`もしくは`std_logic_vector`に変換しておくことをお勧めします．また，すべての信号を保存するため，大規模な回路のときには大きなファイルを作ってしまいます．そのような場合には，大容量ディスクを用意しておく必要があるでしょう．

VCDファイルのフォーマットですが，もともとVerilog HDLから来ているので‘1’,‘0’,‘Z’,‘X’しか扱えません．GHDLでは，VCDファイルに出力する際に，上記のいずれかに変換してしまうようです．

波形を表示させる際にはVCDファイルに対応したビューワを使用します．そのビューワにもフリー・ソフトウェアがいくつかありますが，GHDLの作者は「GTKWave」を推奨しています．筆者も波形を見る場合はこれを使っています．GTKWaveはaptグループのWebサイト（http://www.cs.man.ac.uk/apt/）から入手できます．VCDファイルさえ表示できればよいので，Windowsで動作するツールでも利用できます．

● 商用シミュレータと同等の速度

ここまではすべて，GHDLのマニュアルに示されたコードを使った評価でした．次に，筆者が記述したコードを使って評価してみます．簡単なアップダウン・カウンタ（**リスト1**）とそのテスト・パターン（**リスト2**）を用います．

二つのファイルを一気にアナライズからエラボレート，シミュレーションまでまとめて実行できるように，専用のバッチ・ファイル（**リスト3**）を作ってみました．

実行結果をGTKWaveによる出力波形で示します（**図3**）．

リスト1　簡単なアップダウン・カウンタのVHDLソース・コード(udc16.vhd)

```
library ieee;
use ieee.std_logic_1164.all;

Entity UDC16 is
 Port(CLK,RESET,U_D:in std_logic;
      QA,QB,QC,QD:out std_logic
      );
end UDC16;

Architecture BUDC16 of UDC16 is
function cnv_std(VINT,num:integer) return std_logic_vector is
   variable VRES:std_logic_vector(num-1 downto 0);
   variable tint:integer;
     begin
     tint := VINT;
     for i in 0 to num-1 loop
       if (tint mod 2)=1 then
         VRES(i):='1';
       else
         VRES(i):='0';
       end if;
       tint := tint/2;
     end loop;
     return VRES;
   end cnv_std;

  begin
   process(RESET,CLK)
     variable cint:integer;
     variable Dout:std_logic_vector(3 downto 0);

     begin
      if RESET='0' then
         cint:=0;
      else
        if CLK'event and CLK='1' then
          if U_D='1' then
               cint:= cint +1;
                 if cint=16 then
                   cint:=0;
                 end if;
          else
              cint:= cint -1;
                 if cint<0 then
                   cint:=15;
                 end if;
          end if;
        end if;
      end if;
      Dout:= cnv_std(cint,4);
      QD<=Dout(3);
      QC<=Dout(2);
      QB<=Dout(1);
      QA<=Dout(0);
     end process;
   end BUDC16;
```

リスト2　アップダウン・カウンタのテストベンチ(ts_ud16.vhd)

```
library ieee;
use ieee.std_logic_1164.all;
use work.all;
Entity ts_ud16 is
end ts_ud16;
Architecture test1 of ts_ud16 is
 Component udc16 Port(CLK,RESET,U_D:in std_logic;
                      QA,QB,QC,QD:out std_logic
                      );
 end component;

  signal CLK:std_logic:='0';
                          --set clock initial value
  signal RESET,U_D:std_logic;
  signal QA,QB,QC,QD:std_logic;

Begin
  M1:udc16 port map(CLK,RESET,U_D,QA,QB,QC,QD);
  CLK <= not CLK after 50 ns;  -- define clock
  RESET <='1',
          '0' after 100 ns,
          '1' after 200 ns;
  U_D <= '0',
         '1' after 1500 ns;

end test1;
```

GUIがない分軽いのか，商用のシミュレータと比較しても遜色ない速度でした．

● バッチ・ファイルを活用してデバッグ

GHDLには商用ツールのようなデバッグ環境がありません．GCCのデバッグ・ツールであるGDBが使えるとマニュアルには書いてありますが，筆者の意見としてはGDBを使ってのデバッグはありえないと思っています．なぜならGDBが出力する情報はVHDLに根ざしたものではなく，GHDLというソフトウェアのデバッグのための情報がほとんどだからです．イベントが起こった時点で停止するような機能なら使うかもしれませんが，デバッグ・コマンドで直接VHDLの状況を見ることはできないからです．

ほとんどの設計者は，波形とVHDLソース・コードを見比べながらデバッグを行うことでしょう．そのため，波形ビューワと優れたエディタさえあれば，たいていのデバッグ作業はできてしまいます．あとはコンパイルとシミュレーションを実行してくれるバッチ・ファイルを作れば，デバッグ環境としては十分です．そういう意味では今のままのGHDLでもデバッグには差し支えないといえるでしょう．

GHDLにGUIは備わっていません．そのため，バッチ・ファイルを使って，シミュレーションとデバッグを交互に繰り返すような使いかたになります．米国Mentor Graphics社のModelSimのようなGUIに慣れた方には不満もあるでしょうが，バッチ・ファイルを組むのがそれほど苦にならない方なら使いやすいでしょう．

● FPGA設計用や学習用に活用できる

GHDLはVITALもサポートしています．現段階ではVITALのレベル1までのサポートで，バックアノテーションの機能も備わっています．しかし，商用ツールのように，ネットリストに遅延情報をバックアノテーションするのは

リスト3　バッチ・ファイル（make_sim）

```
#!/bin/bash
if [ ! -x work ]; then
mkdir work
else
ghdl --clean --workdir=work
fi
ghdl -i --workdir=work  ./*.vhd
ghdl -m --workdir=work  ts_ud16
ghdl -r   ts_ud16 --stop-time=3000ns --vcd=wave.out
gtkwave wave.out
```

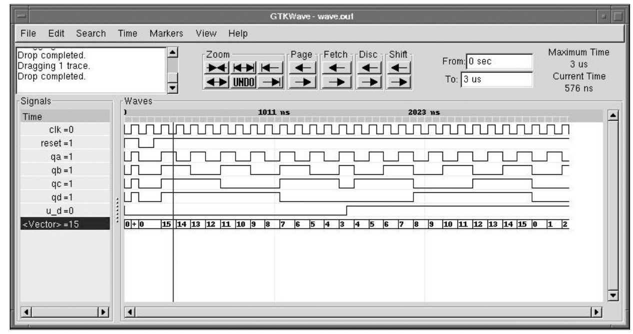


図3　GTKWaveによる波形の表示

今のところできないようです．まだ，ASICの設計に用いることは難しいでしょう．

それでもFPGAの設計には使えそうです．ただ，残念なことに，今はLinux版しかありません．Linuxで稼働するFPGAのツールは，Windows版と比べると少ないことが問題になるかもしれません．

まだベータ版という位置づけですが，学習用としては申し分ないものだと思います．実際に使用している学校もあると聞きます．

たけもと・さとる
(有)サイバーラボ

参考・引用*文献

(1) GHDLのホームページ，http://ghdl.free.fr/
(2) GHDLのマニュアルの和訳，http://www.cyber-labo.co.jp/ghdl_guide/top.html
(3) GTKwaveのホームページ，http://www.cs.man.ac.uk/apt/tools/gtkwave
(4) 竹本 悟；「Icarus Verilog」＋「IVI」の使い勝手を試す，Design Wave Magazine，2005年5月号.

＜筆者プロフィール＞
竹本 悟．1952年神戸市生まれ．高校時代にラジオやアンプを作っていた世代ですが，いつの間にかHDL設計を飯の種にするようになっていました．サラリーマンを20年，二つの会社に勤めましたが，やっぱりやってみたくて独立しました．ときどきCQ出版でもセミナの講師を務めています．趣味は碁，つり，キャンプなどいろいろやりますが，回路を考えているときがいちばん楽しいと思っています．もちろん熱烈な阪神ファンです．